

Encrypted TCP chat using RSA and AES algorithm

Krzysztof Jakub Szala

Bachelor's Thesis

April 2014

Degree Programme in Information Technology
Technology, communication and transport



JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES



| | | |
|--|--|---|
| Author(s) SZALA Krzysztof Jakub | Type of publication Bachelor's Thesis | Date 23.04.2014 |
| | Pages 46 | Language English |
| | | Permission for web publication (X) |
| Title ENCRYPTED TCP CHAT USING RSA AND AES ALGORITHM | | |
| Degree Programme Information Technology | | |
| Tutor(s) MIESKOLAINEN, Matti | | |
| Assigned by Descom Oy | | |
| <p>Abstract</p> <p>Secure Communication Application is a standalone project which can be used to secure communication between two computers running on Windows. The purpose of the project was to create an application which can be easily divided in-to parts, which allows replacing the user interface by a different one. The project was realized under the supervision of Descom oy.</p> <p>This thesis demonstrates the process of creating the core of the application responsible for communication over TCP protocol between two computers and the user interface. Secure communication was the most important part of it.</p> <p>The first part explains the reasons and motivations to create such an application. The goals are also discussed. The following chapter presents the development environment and important theory. Next, the way of designing and developing this application is discussed. Later the way how to replace a user interface with another is demonstrated and finally, the application is tested and the results and conclusion are presented.</p> <p>The result of the thesis is functional software able to ensure secure communication between two computers. The programs were shown with success to the project's supervisor. Nevertheless, further development would be required to make the application more secure and smooth</p> | | |
| Keywords C++, Java, message, chat, sending, receiving, encryption, AES, RSA, OpenSSL, TCP | | |
| Miscellaneous | | |

CONTENT

| | |
|---|----|
| CONTENT | 2 |
| ACRONYMS | 4 |
| FIGURES | 5 |
| 2 OBJECTIVE OF THE THESIS | 8 |
| 3 DEVELOPMENT ENVIRONMENT | 9 |
| 3.1 C/C++ | 9 |
| 3.2 JAVA | 9 |
| 3.3 OPENSSL | 9 |
| 4 THEORY | 11 |
| 4.1 CIPHERING ALGORITHMS..... | 11 |
| 4.2 RSA OVERVIEW..... | 12 |
| 4.3 RSA DETAILS..... | 12 |
| 4.4 AES OVERVIEW..... | 14 |
| 4.5 AES DETAILS | 14 |
| 4.6 HMAC | 17 |
| 4.7 TCP PROTOCOL | 18 |
| 5 DESIGN AND IMPLEMENTATIONS | 19 |
| 5.1 CONCEPT | 19 |
| 5.2 COMMUNICATION LAYER | 26 |
| 5.2.1 Winsock | 26 |
| 5.2.2 Parameters and requirements | 29 |
| 5.3 CIPHERING LAYER..... | 30 |
| 5.3.1 Creating/Reading RSA key from file | 30 |
| 5.3.2 Creating AES key | 31 |
| 5.3.3 AES key exchange | 32 |
| 5.3.4 Encrypting/decrypting AES sessions key using RSA algorithm .. | 32 |
| 5.3.5 Using AES algorithm..... | 33 |
| 5.3.6 HMAC | 34 |
| 5.4 USER INTERFACE LAYER..... | 36 |
| 5.4.1 Sockets..... | 36 |
| 5.4.2 Data taken from user | 36 |
| 5.4.3 How to change UI | 38 |
| 6 INSTALLATION | 40 |

| | |
|-------------------------------|----|
| 7 TESTING | 41 |
| 8 FURTHER DEVELOPMENT..... | 44 |
| 8.1 WHAT HAS BEEN DONE | 44 |
| 8.2 WHAT SHOULD BE DONE | 44 |
| 9 CONCLUSION | 45 |
| REFERENCES | 46 |

ACRONYMS

ACK - Acknowledgment

AES – Advanced Encryption Standard

HMAC - Hash-based message authentication code

IP – Internet protocol

IV – Initialization vector

JDK - Java Developing Kit

LAN – Local area network

MAC - Message Authentication Code

UI – User interface

SYN - Synchronize

TCP – Transmission Control Protocol

VS – Visual Studio

FIGURES

| | |
|---|----|
| FIGURE 1 Amount of data in internet (Silicon angle). | 6 |
| FIGURE 2 High level architecture message cycle. | 21 |
| FIGURE 3 High level architecture with port descriptions. | 22 |
| FIGURE 4 Example scenarios for application. | 23 |
| FIGURE 5 Example configurations. | 37 |
| FIGURE 6 Example message exchange. | 38 |
| FIGURE 7 How RawCap is working. | 42 |
| FIGURE 8 Message sent from UI to C++ application. | 42 |
| FIGURE 9 Message sent from one C++ application to another one. | 43 |
| FIGURE 10 Message sent from C++ application to UI. | 43 |

1 INTRODUCTION

“We live in a society that is awash with information, but few of us really understand what information is.” (Floridi L. 2010)

Nowadays people need to communicate with each other all the time. Communication often takes place between people who are far from each other, so they use the Internet to that. They are exchanging information with significant value. This fact leads to the statement that the message that they are exchanging should remain secret for other parties who are not authorized.

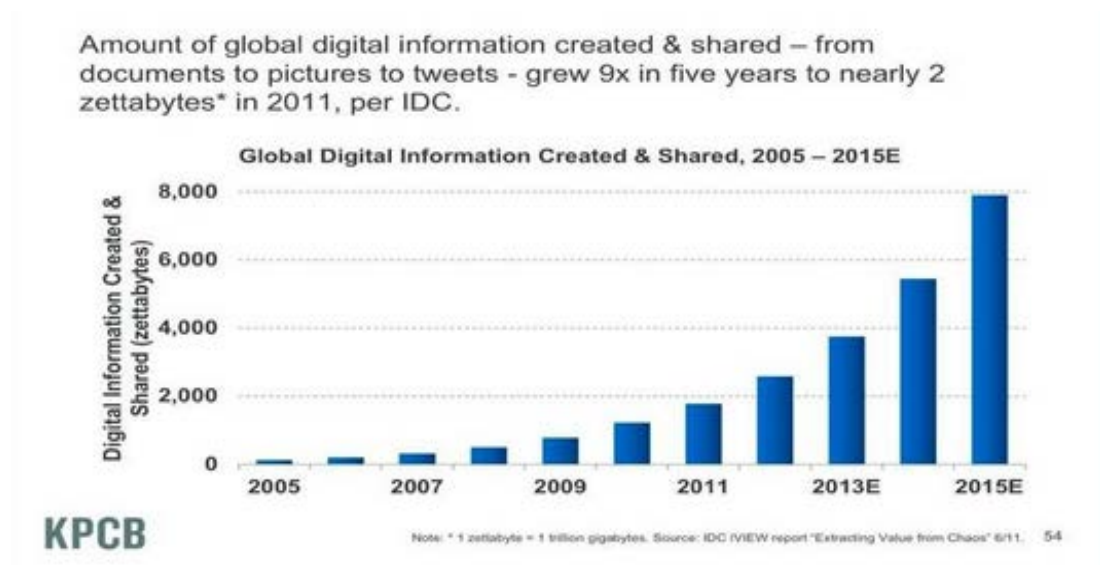


FIGURE 1 Amount of data on the internet (Silicon angle).

Security is nowadays one of the most important issues in a network. Every company should have their own application which enables their workers to communicate securely and freely. Besides messages, people exchange files which should also be encrypted.

To achieve secure communication application allowing users to exchange message via Internet should have implemented cryptographic algorithms which should be checked, tested and reliable since people exchanging a lot of messages, encryption algorithms should work fast.

According to everything what was mentioned previously, it was decided to an create application with implemented cryptographic algorithms that meets the previously mentioned goals. The name of the application is Secure Communication Application (further referred to as SecureCA).

2 OBJECTIVE OF THE THESIS

As mentioned in the previous chapter the application should be:

- Fast
- Working over TCP protocol
- Reliable
- Allow to communicate between two users
- Work on computer with Windows 7 x64 operating system
- Implementation should allow to easily change user interface in future

To provide the above requirements it was decided to use cryptographic function:

- RSA algorithm
- Advanced Encryption Standard
- Hash-based message authentication code

The project assumed providing communication only between two users at a time. Communication should take place via two different channels, one allowing to receive message, the other one to send them. Channels are working independently one from another. It was also assumed that before the establishment of communication, users exchange their public keys, although the application can create a new key pair for user, but after that the keys should be exchange again and application needs to be restarted again.

Ensuring communication with multiple users at the same time and certification authority should be implemented in the future. Each time the application is started it should create a new session key. To simplify establishing a connection it was also assumed that each user know that communication will take place, and run his own server before other user starts to try connect to that server.

3 DEVELOPMENT ENVIRONMENT

The application was developed under Windows 7 x64 operating system.

3.1 C/C++

The core of the application responsible for implementations of ciphering algorithm and communication via network was written using C++ programming language in Microsoft Visual Studio 2012. This language was chosen because it is “fast”, efficiency and compatible with library, which was used to provide implementation of cipher algorithms, written using C language.

When installing Microsoft Visual Studio 2012 all needed libraries are installed so that there is no needed to install anything else. Microsoft Visual Studio 2012 license is also needed. Instead of VS 2012 NetBeans can be used.

3.2 Java

The user interface was written using Java programming language in NetBeans which can be found on the following webpage:

<https://netbeans.org/downloads/>

Before installing NetBeans JDK needed to be installed. The latest version of JDK can be downloaded on the following webpage:

<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

3.3 OpenSSL

OpenSSL library was used to provide encrypting algorithms. In this project Win32 OpenSSL v 1.0.1 f was used. This library can be downloaded from following webpage:

<http://slproweb.com/products/Win32OpenSSL.html>

To use OpenSSL library in C++ project following needed to be done in project properties:

- C/C++ tab in Additional Include Directories set path to folder with OpenSSL library, subfolder include(default: C:\OpenSSL-Win32\include)
- Linker tab in Additional Library Directories set path to folder with OpenSSL library, subfolder lib, subfolder VC(default: C:\OpenSSL-Win32\lib\VC)
- In Linker tab, subtab Input in Additional Dependencies set name of library that should be used in project(in this project was used: libeay32MDd.lib, ssleay32MDd.lib)
- The function from OpenSSL library that was used in that project will be described later on in this thesis.

4 THEORY

To create ciphering application it is very important to understand first algorithm that are used in that kind of program. If theory placed below is not enough it is recommended to read Fips 197 about AES and PKCS #1 about RSA.

4.1 Ciphering algorithms.

Before starting to develop that project, there were very important decisions to be made. The algorithm was chosen carefully according to information about security and reliable.

As asymmetric algorithm to encrypt session key RSA algorithm was chosen. AES was decided to be symmetric algorithm responsible for encrypting and decrypting messages. HMAC with SHA-256 as hash function was chosen to be solutions for authenticating messages.

An important part of that application is also the communication protocol which allows hosts to send information via public Internet.

4.2 RSA overview.

RSA is an asymmetric encrypting algorithm normally used to send a session key between users and which will be used in the future by users in ciphering with the use of symmetric algorithms. This algorithm is not used to cipher communication because it needs a great deal of resources and time to encrypt and decrypt data. RSA uses two keys in ciphering, public key and private key. User provides the public key to other users and protects and keeps his/her own private key in secret. It is strongly recommended to use at least 2048 bits long key because shorter keys are considered not to be secure. Also, public exponents should be as large as possible to secure users from attack against small public exponents. (RSA PKCS)

The strength of this algorithm lies in two mathematical problems:

- The problem of factoring large numbers
- RSA problem

4.3 RSA details.

To generate RSA key pair this algorithm had to be used:

1. Chose randomly two large prime numbers p and q .
2. Solve $n = p \cdot q$
3. Solve Euler function value for n : $\phi(n) = (p-1) \cdot (q-1)$
4. Chose number e such as $1 < e < \phi(n)$ relatively prime with $\phi(n)$
5. Solve $d = e^{-1} \bmod \phi(n)$

Public key is defined as number pair (n, e) while private key is defined as pair (n, d)

To encrypt with RSA algorithm message have to be divide in to m_i blocks of value not larger than n and then cipher with pattern:

$$c_i = m_i^e \bmod n$$

To decrypt with RSA algorithm every c_i block had to be transform like this:

$$m_i = c_i^d \bmod n$$

Until March 2nd 2014 the larger key that was decomposed into prime factors 768-bit length key, RSA is considered to be secure ciphering algorithms nowadays.

4.4 AES overview.

AES is a symmetric encrypting algorithm normally used to encrypt data with one the same key for encryption and decryption which works in various modes. For this application, counter mode was chosen. The algorithm is based on Rijandel algorithm, a symmetric block cipher able to transform 128-bit long data blocks. The key can have three different lengths: 128, 192 and 256 bits. Depending on the key length, the algorithm consists of 10, 12 or 14 rounds. (Fips 197)

4.5 AES details

The algorithm operates on a 4x4 column-major order matrix of bytes called “state” arranged as follows:

$$\begin{bmatrix} \text{byte}_0 & \text{byte}_4 & \text{byte}_8 & \text{byte}_{12} \\ \text{byte}_1 & \text{byte}_5 & \text{byte}_9 & \text{byte}_{13} \\ \text{byte}_2 & \text{byte}_6 & \text{byte}_{10} & \text{byte}_{14} \\ \text{byte}_3 & \text{byte}_7 & \text{byte}_{11} & \text{byte}_{15} \end{bmatrix}$$

All operations in AES are byte-based. The state consists of 128 bits which are equal to 16 bytes

High-level AES architecture:

1. AddRoundKey
2. For each of rounds(10, 12 or 14 times):
 - 2.1. SubBytes
 - 2.2. ShiftRows
 - 2.3. MixColumns
 - 2.4. AddRoundKey
3. After last round “state” is returned as ciphered text.

AddRoundKey - function that XORs the round key with the “state”

SubBytes - function that substitute on each byte with using of one S-box (predefined 16x16 table)

ShiftRows - cyclically shifts the elements of i-th row i elements to the left for encryption and right for decryption as it is shown below:

$$\begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \Rightarrow \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \end{bmatrix}$$

MixColumns – in “state” replaces each byte of a column by a function of all the bytes in the same column. Function for each row is described below:

First row:

$$s'_{0,j} = (0x02 \times s_{0,j}) \otimes (0x03 \times s_{1,j}) \otimes s_{2,j} \otimes s_{3,j}$$

Second row:

$$s'_{1,j} = s_{0,j} \otimes (0x02 \times s_{1,j}) \otimes (0x03 \times s_{2,j}) \otimes s_{3,j}$$

Third row:

$$s'_{2,j} = s_{0,j} \otimes s_{1,j} \otimes (0x02 \times s_{2,j}) \otimes (0x03 \times s_{3,j})$$

Fourth row:

$$s'_{3,j} = (0x03 \times s_{0,j}) \otimes s_{1,j} \otimes s_{2,j} \otimes (0x02 \times s_{3,j})$$

Summarize each byte in a column is replaced by two times that byte, plus three times next byte, plus the byte that left. It should be mentioned here also that next to the byte that is in the last row is a byte in the first row.

Round key are created base on AES key. First key is arranged as it is shown below:

$$\begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix} \Rightarrow [w_0 \quad w_1 \quad w_2 \quad w_3]$$

Where w is called word and it consists of four bytes from key. Based on those four words next four are created and so on until 40 words are obtained besides starting four. Below is the algorithm how to get next four words:

$$w_{i+4} = w_i \otimes g(w_{i+3})$$

$$w_{i+5} = w_{i+4} \otimes w_{i+1}$$

$$w_{i+6} = w_{i+5} \otimes w_{i+2}$$

$$w_{i+7} = w_{i+6} \otimes w_{i+3}$$

Where g function consists of following steps:

1. One-byte left circular rotation on the argument.
2. SubBytes explained on previous page.
3. XOR obtained bytes with round constant.

Where round constant is created this way:

$$Rcon[i] = (RC[i], 0x00, 0x00, 0x00)$$

$$RC[1] = 0x01$$

$$RC[j] = 0x02 \times RC[j-1]$$

4.6 HMAC

HMAC is MAC code with mixed secret key. Thanks to that ensures both the protection of the integrity and authenticity of data. Standard MAC code provides integrity protection, however, it may be subject to adulteration if it is not protected by an additional mechanism to protect its authenticity. To protect the integrity and authenticity HMAC was created. As it was mentioned before HMAC adds secret key to each MAC message:

$$HMAC_K(m) = h((K \oplus opad) \| h((K \oplus ipad) \| m))$$

Where opad and ipad are fixed complementary values, m is text for which HMAC is created and K is secret key. To create right HMAC code the key is needed, which ensures that the data are protected. In this application HMAC based on SHA-256 cryptographic hash function was used. (RFC 2014)

4.7 TCP protocol

TCP protocol was chosen to be the internet layer on which communication will be built and established. This protocol provides reliable, ordered and error-checked delivery of messages between hosts via LAN, intranet or public Internet. It belongs to transport layer of TCP/IP suite and communication services between application program and IP.

TCP works in client-server mode. Server is waiting for connection on specified port, while client is trying to initialize the connection. This protocol guaranteed that all messages will be delivered in correct order and without duplications. This ensures a reliable connection at the expense of greater overhead in the form of a header and a larger number of packets sent.

To establish a connection three-way handshake procedure is used, like this:

1. Host A send to host B SYN segment with information about the lower end of the sequence numbers used to number of segments sent by host A.
2. Host B replies with SYN-ACK segment which contains host B sequence number and acknowledgment number which is host A sequence number incremented by 1.
3. Host A send to host B ACK segment with acknowledge number which is host B sequence number incremented by 1.

After that three steps connection is considered to be established and normal communication can be sent. After one of host receive packets from another one it this host should reply with ACK segment which contains sequence number of previously received data. (RFC 793)

5 DESIGN AND IMPLEMENTATIONS

5.1 Concept

The idea for the project was to create an application that provides secure communication over TCP protocol and the user of which interface can be easily changed into a different one, even implemented in a different language than Java programming language.

To provide the aforementioned functionality, the project was divided into three layers such as communication, ciphering and user interface layer.

Communication and ciphering layer are integrated into one application which can be launched alone without user interface.

The ciphering layer can be easily extracted from that project and used to create application to ciphering files, or to provide ciphering in another operation system than the Windows. It provides implementations of symmetric and asymmetric algorithms to encrypt and decrypt data.

User does not have to interact with the ciphering or communication layer. It is only needed to input the IP address and the port number with which communication should be established

C++ language was chosen to be the language with the use of which the application core responsible for communication over TCP protocol and ciphering is to be written. To create a user interface using Java programming language was chosen due to the easiest method in implementing graphic elements.

It is recommended to use a ready library to provide an encryption algorithm and not to implement it by ourselves because it is considered to be more secure. All these kinds of libraries are public and it was tested many times and proved to be secure and well implemented. Therefore it was decided to use OpenSSL as library with encrypting function.

Communication between Java and C++ application was decided to be implemented with usage of sockets on port 8080 and 8081 and IP address 127.0.0.1. That IP address is a loopback address in the Windows operating system, and allows to send information between the ports of the same local machine.

A server implemented in Java uses port 8080 to receive data from C++ application and the client is using port 8081 to send data to C++ application. Decrypting C++ application has a client which sends data obtained from a partner computer working on port 8080. C++ application encrypting data has server listening on port 8081 for data from Java application. As it was mentioned earlier user has to input port number of partner's server in user interface and partner's IP address and also port for his own server.

The application and all files should be placed in directory: "C:/SecureCA/". Otherwise, program will not run properly.

Below is the scenario for the application:

User on PC1 types some data into the user interface application and decides to send it. Data is sent via port number 8081 to the ciphering application which encrypts them and sends via port the number of which was earlier defined when the application was started and on an IP address which also was earlier defined. C++ application which is runs on PC2 whose IP address is the same as the one defined on PC1 user interface application receives data from PC1 on port the number of which was defined earlier in the user interface. They are decrypted and sent via port 8080 to the user interface which is receiving decrypted data on port number 8080, and finally data are shown on user interface for user. (See Figure: 1. 2. 3.)

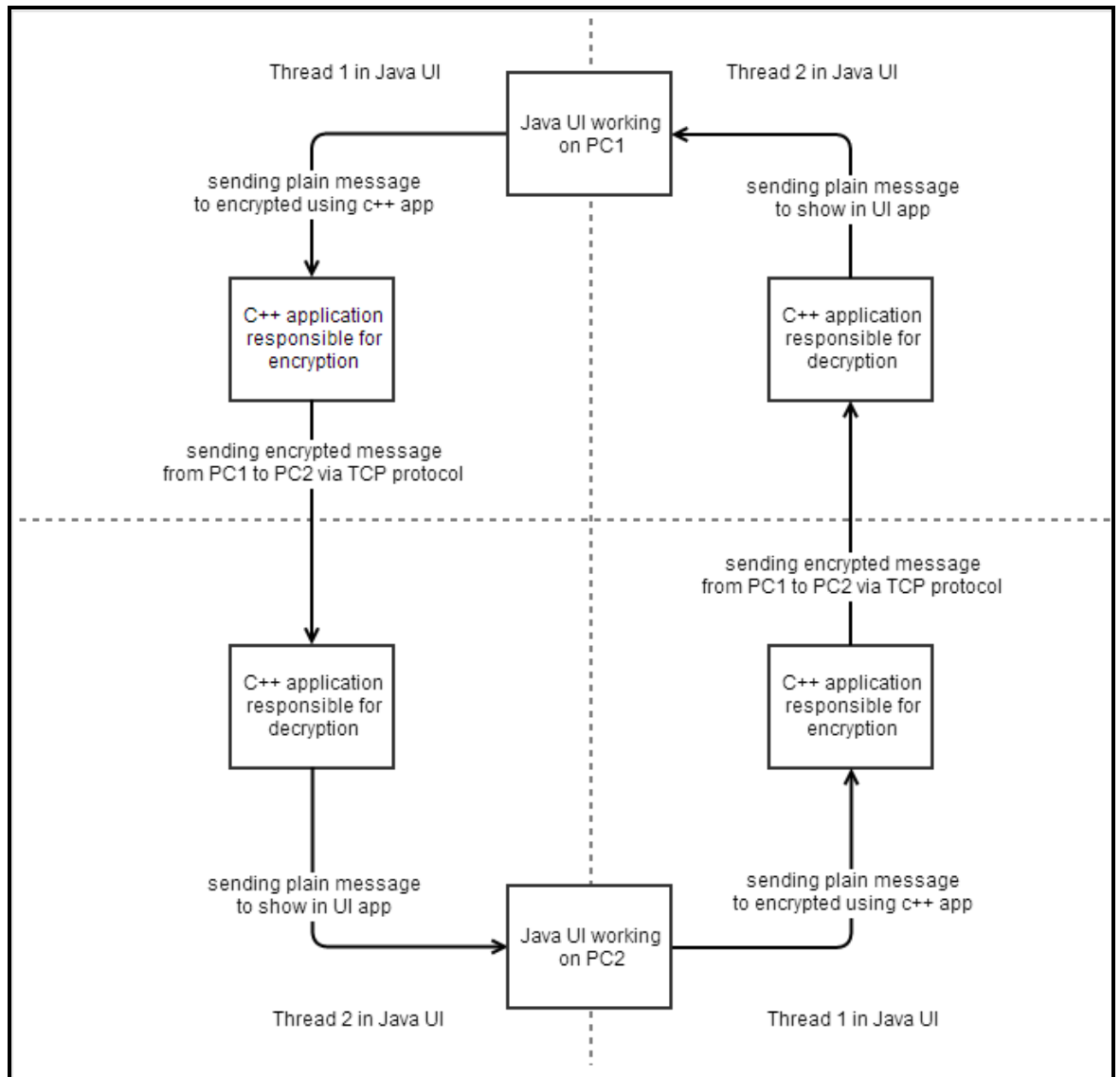


FIGURE 2 High level architecture message cycle.

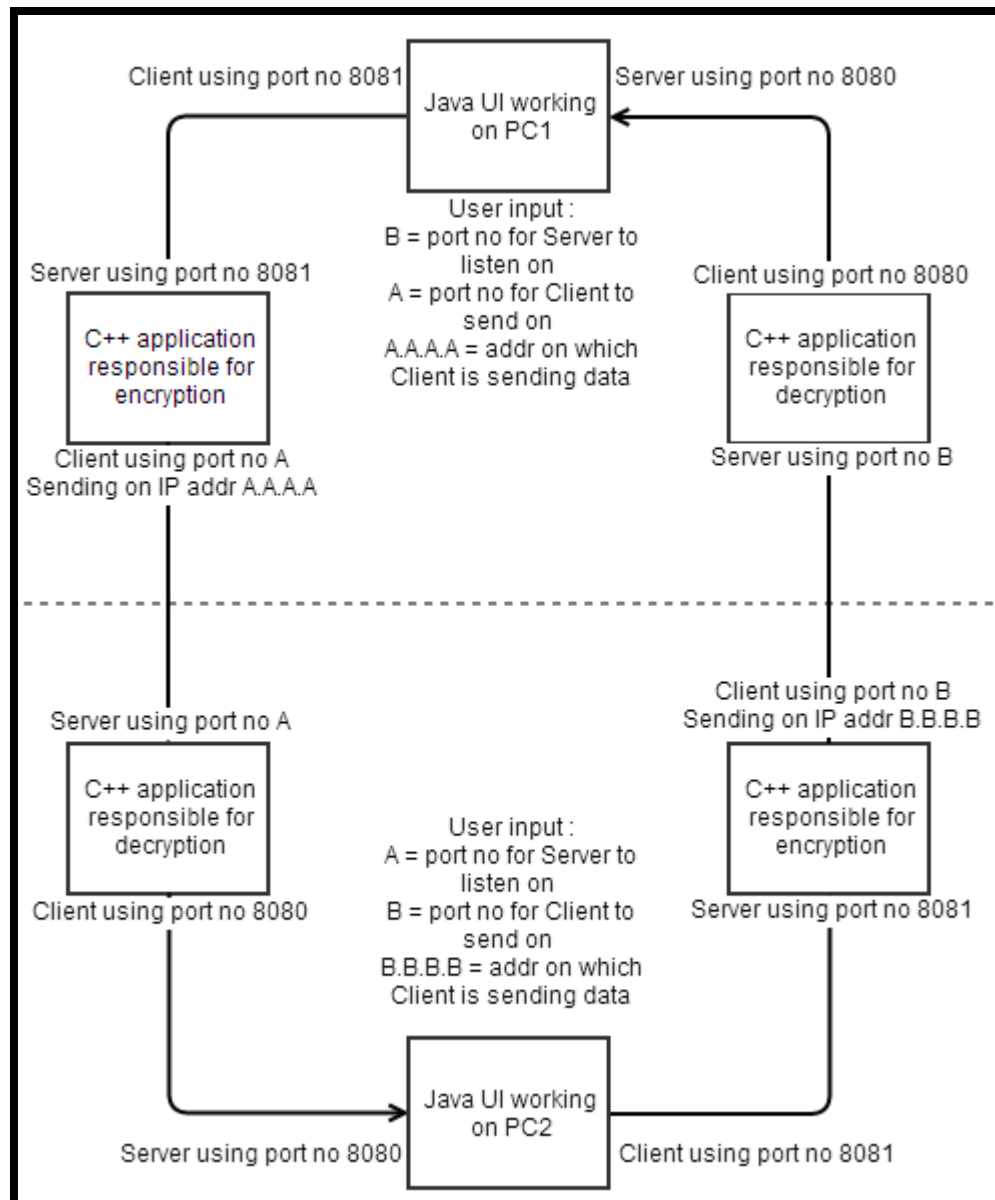


FIGURE 3 High level architecture with port descriptions.

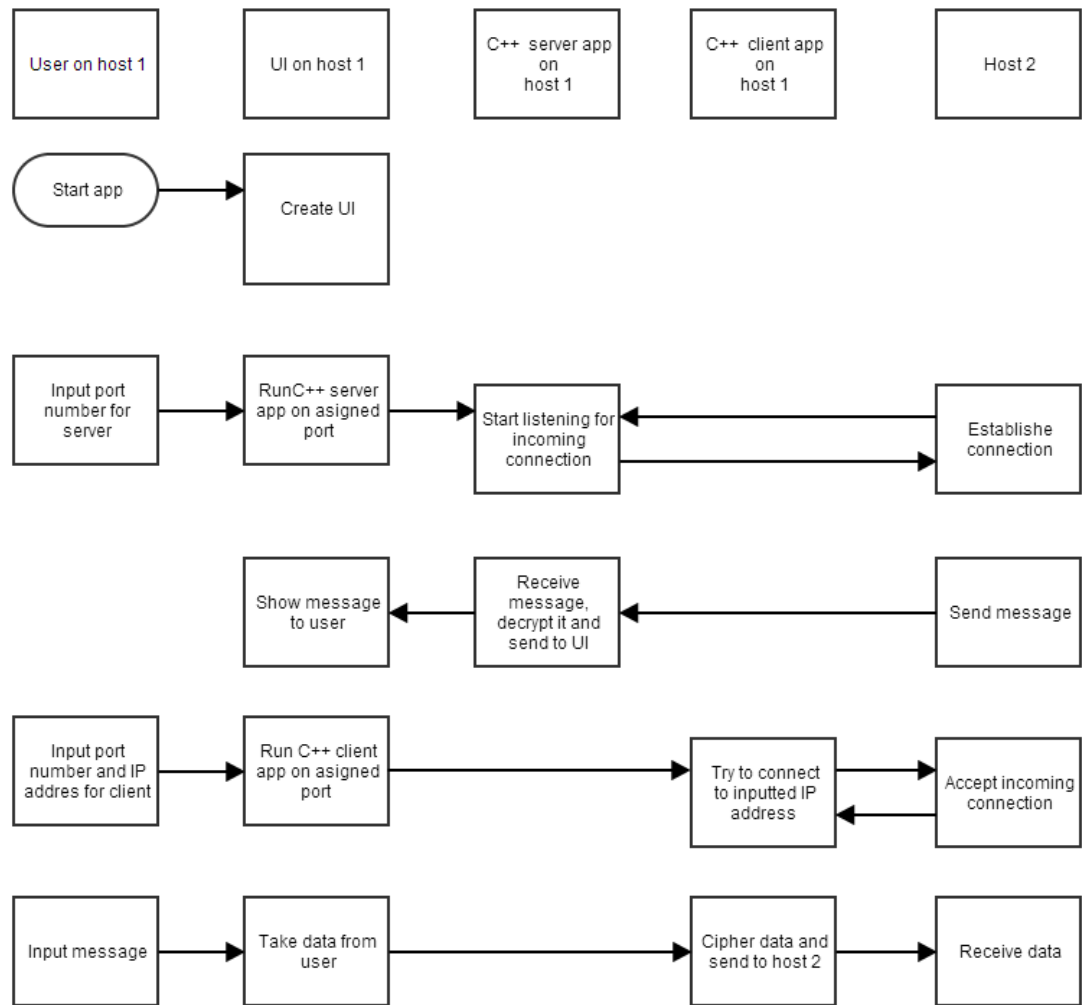


FIGURE 4 Example scenarios for application.

Below is the scenario for the C++ "SecureCASServer":

1. Start application.
2. Establish the connection with UI
3. Start to listen for incoming connection from "SecureCA" working on another host.
4. Accept the incoming connection from another host.
5. Receive encrypted AES key received from another host.
6. Decrypt AES key.
7. Receive encrypted IV from another host
8. Decrypt IV received from another host
9. Initialize data for AES cipher.
10. Start to listen for incoming message from the other host.
11. When receive any message:
 - a. Store it as the "hMACmessage".
 - b. Store next incoming message as "EncryptedMessage".
 - c. Create the hash from the "EncryptedMessage".
 - d. Compare created hash with the received one.
 - i. If they are the same decrypt the "EncryptedMessage", and send to UI
 - ii. Otherwise do nothing
 - e. Still listen for incoming message
12. If the connection with another host is lost, turn off the application

Below is the scenario for the C++ "SecureCAClient":

1. Start application.
2. Start to listen for incoming connection from UI.
3. Accept the incoming connection from UI.
4. Establish connection with "SecureCA" working on another host.
5. Generate the AES key and the IV.
6. Encrypt AES key.
7. Send encrypted AES key to another host.
8. Encrypt IV.
9. Send encrypted IV to another host
10. Initialize data for AES cipher.
11. Start to listen for incoming message from the UI.
12. When receive any message:
 - a. Encrypt that message using AES algorithm and store as "EncryptedMessage".
 - b. Create the hash from the "EncryptedMessage".
 - c. Send created hash to another host.
 - d. Send "EncryptedMessage" message to another host.
 - e. Still listen for incoming message
13. If the connection with UI is lost, turn off the application.

5.2 Communication layer

5.2.1 Winsock

Communication between two hosts is implemented with the help of Winsock. It is a technical specification that defines how to access the network services from Windows operating system. Microsoft deliver the library needed to implement that communication. In that application “windows”, “winsock2” and “ws2tcpip” are used. Those libraries provide us with the basic function and data structure necessary to establish the connection.

Server side needs two sockets, one for listening for incoming connection. The other one should be assigned after accepting the incoming connection which was established. After that there is no need to listen to the socket anymore, only the second socket needs to send data from client received.

The following code shows data needed to initialize connection based on Winsock server side:

```
WSADATA wsaData;
SOCKET ListenSocket;
SOCKET ClientSocket;
struct addrinfo *result;
struct addrinfo hints;
```

The following code shows data needed to established connection based on Winsock server side:

```
WSAStartup(MAKEWORD(2,2), & getWsaData());
ZeroMemory(&hints, sizeof(hints));
hints.ai_family = AF_INET;
hints.ai_socktype = SOCK_STREAM;
hints.ai_protocol = IPPROTO_TCP;
```

```
hints.ai_flags = AI_PASSIVE;
```

The following code shows process of creating, listening and accepting connection on server side:

Resolve the server address and port

```
getaddrinfo(NULL, portNumber, &hints, &result);
```

Create a SOCKET for connecting to server

```
setListenSocket(socket(result->ai_family, result->ai_socktype, result->ai_protocol));
```

Setup the TCP listening socket

```
bind(getListenSocket(), result->ai_addr, (int)result->ai_addrlen);
```

Listen for incoming connection

```
listen(getListenSocket(), SOMAXCONN);
```

Accept a client socket

```
setClientSocket(accept(getListenSocket(), NULL, NULL));
```

The client side needs only one socket which is used to connect to server, and after the connection is established, this socket is used to send and receive data from server.

The following code shows the data needed to initialize connection based on Winsock client side:

```
WSADATA wsaData;
```

```
SOCKET ConnectSocket;
```

```
struct addrinfo *result;
```

```
struct addrinfo *ptr;
```

```
struct addrinfo hints;
```

The following code shows data needed to established connection based on Winsock client side:

```
WSAStartup(MAKEWORD(2,2), &getWsaData());

ZeroMemory( &hints, sizeof(hints) );

hints.ai_family = AF_UNSPEC;

hints.ai_socktype = SOCK_STREAM;

hints.ai_protocol = IPPROTO_TCP;
```

The following code shows process of creating and connection to server on client side:

Resolve the server address and port

```
getaddrinfo(getIPAddress(),getPortNumber(), &hints, &result);
```

Attempt to connect to an address until one succeeds

```
for(ptr=result; ptr != NULL ;ptr=ptr->ai_next) {
```

Create a SOCKET for connecting to server

```
setConnectSocket(socket(ptr->ai_family, ptr->ai_socktype, ptr->ai_protocol));
```

Connect to server.

```
connect(getConnectSocket(), ptr->ai_addr, (int)ptr->ai_addrlen);

}
```

After establishing the connection data are exchanged between server and client with the usage of two functions send() and recv(). Data sent between users in SecureCA cannot be longer than 256 bytes which allow user to send at once 256 chars.

The following code shows function used to send and receive data:

Sending and receiving data on client side

```
send(ConnectSocket, sendbuf, (int) strlen(sendbuf), 0);

recv(ConnectSocket, recvbuf, recvbuflen, 0);
```

Sending and receiving data on server side

```
recv(ClientSocket, recvbuf, recvbuflen, 0);
```

```
send(ClientSocket, recvbuf, iResult, 0);
```

“Send” and “recv” function requires 4 variables as input. The first is a descriptor that identifies the connected socket which should send/receive data. The second is a pointer to the buffer to send/receive data. The third is the length in bytes, of the sending data or the size of buffer for storing the received data. The fourth is not used in that program and it is a set of flags that influences the behavior of those functions. (MSDN Send, MSDN Recv)

5.2.2 Parameters and requirements

The implementation of communication layer requires from user to deliver two parameters to set up the client side and one parameter for the server layer.

For the client the users need to input port number on which communication will be established. The port number should be higher than 1023 and lower than 65535. The second parameter is the IP address of the host with which user want to communicate. It should be 32-bit number consisting of 4 octets in dot-decimal notation compatible with Internet Protocol Version 4, in example 191.168.1.1. (RFC 3330, RFC 6335)

For the server user need to input only one parameter which is the port number on which the server will be listening for incoming connection. The port number should be a figure between 1023 and 65535. (RFC 6335)

5.3 Ciphering layer

Ciphering layer was written using C++ programming language with usage of OpenSSL library which provide basic cryptographic function. To enable RSA and AES function it was needed to include in project file such as “rand.h”, “rsa.h”, “aes.h”, “hmac.h”, “engine.h”, “applink.c” and “pem.h”. This layer consist of two independent programs, the first (further referred to as SecureCASServer) is responsible for receiving data, decrypting and sending to user interface, the econd (further referred to as SecureCAClient) is responsible for receiving data from user interface, encrypting and sending to person with which the user wants to communicate.

5.3.1 Creating/Reading RSA key from file

This program allows user to create RSA key pair or read it from file. The key length is 4096 bits and the exponential used to create key is 65537. The key is stored in RSA* data type. If while the application was started there were no RSA key pairs in the folder where the application is placed, they are created and saved in the same folder where the application is, however, is required to restart the application to work properly. The following code listing shows functions which are used to generate, read and write to file RSA key. SecureCASServer reads another host public key, while SecureCAClient reads the host on which the private key from file is running.

```
RSA_generate_key(RSA_KEYLENGTH, RSA_E, NULL, NULL);
PEM_write_RSAPrivateKey(filePrivateRSAKey, this->thisRSAKey, NULL, NULL, 0, NULL, NULL);
PEM_write_RSAPublicKey(filePublicRSAKey, this->thisRSAKey);
PEM_read_RSAPrivateKey(filePrivateRSAKey, &this->thisRSAKey, NULL, NULL);
PEM_read_RSAPublicKey(filePublicRSAKey, &this->otherRSAKey, NULL, NULL);
```

RSA_generate_key function needs to provide 4 parameters. The first is the length of the key, the second exponent number, and the third and fourth are not used in that program. (OpenSSL RSA)

PEM are a family of functions provided by OpenSSL to operate with a file. Read function requires 4 parameters. The first one is a pointer to file from which the key should be read, the second pointer to a variable in which the key should be written, the third is a callback and the fourth is a pass phrase to file. The last two are not used in that program. In PEM_write_RSAPublicKey the first parameter is a pointer to which file key should be written. The second is a pointer to data from which the key should be read. In PEM_writeRSAPrivateKey the first and the second parameter are the same as in PEM_write_RSAPublicKey function. The last five arguments refer to secure file with a password and they are not used in that program. (OpenSSL PEM)

5.3.2 Creating AES key

AES key is created for every new communication session for each channel so there are two keys independent for each other. It is 128-bit long and is created with usage of random bits. Because AES is used in counter mode there is also initialization vector needed to be created which is 16-byte long random bytes. The key is stored in a data structure called AES_KEY. SecureCAClient is responsible for generating AES key while SecureCA Server is receiving that key from another host thanks to SecureCAClient. The following code listing shows the functions which are used to generate, assign and store AES key.

```
AES_KEY aes_key;

RAND_bytes(this->thisAESKey, AES_KEYLENGTH/8)

RAND_bytes(this->thisAESIV, 16)

AES_set_encrypt_key(thisAESKey(), AES_KEYLENGTH, &aes_key);
```

RAND_bytes is a function from OpenSSL library which generates random bytes. Two arguments are needed as input for that function. The first is a pointer to char array in which the data should be stored and the second is the number of bytes that should be generated. (OpenSSL RAND)

AES_set_encrypt_key is a function that generates round keys for AES algorithm and saves them to AES_KEY structure. The function requires three

arguments as input. The first is a pointer to key based on which round keys are generated. The second is the length of the key in bits, this variable can have three different values that are 128, 192, 156 bits. The third argument is a pointer to AES_KEY structure which stores round keys. (Fossies AES)

5.3.3 AES key exchange

AES key are exchanged between hosts right after the connection is established with the usage of RSA algorithm. Initialization vectors are also sent, because they are needed for the correct working of SecureCA. Encrypted key and IV send via TCP protocol, from SecureCAClient working on the first host to SecureCASServer working on the second host, and they are visible only for them.

5.3.4 Encrypting/decrypting AES sessions key using RSA algorithm

To ensure that AES key and IV are not visible for potential attacker who can monitor communication, it was decided to encrypt them with the usage of RSA algorithm. SecureCAClient working on the first host encrypts AES key and IV with RSA public key of the second host and SecureCASServer working on the second host decrypts the received key and the IV with the second host private key.

The following code listing shows the functions which are used to encrypt and decrypt AES key and IV.

```
RSA_public_encrypt(AES_KEYLENGTH/8, ciphering->getThisAESKey(), msgEncrypredAESKey,
ciphering->getOtherRSAKey(), RSA_PKCS1_PADDING);
```

```
RSA_public_encrypt(16, ciphering->getThisAESIV(), msgEncrypredAESKey, ciphering-
>getOtherRSAKey(), RSA_PKCS1_PADDING);
```

```
RSA_private_decrypt(server->iResult,(unsigned char *) msgEncrypredAESKey,(unsigned char *)
msgOtherAESKey, ciphering->getThisRSAKey(), RSA_PKCS1_PADDING);
```

```
RSA_private_decrypt(server->iResult,(unsigned char *) msgEncrypredAESKey,(unsigned char *)
msgOtherAESIV, ciphering->getThisRSAKey(), RSA_PKCS1_PADDING);
```

RSA_private_decrypt is a function which decrypts data previously encrypted using a paired public key and requires 5 arguments. The first is length of decrypting data in bytes. The second is a pointer to char array which should be decrypted. The third is a pointer to char array to which the decrypted data should be written. The fourth is pointer to RSA structure which stores private key. The fifth is the number of padding which should be used. (OpenSSL RSA)

RSA_public_encrypt is a function which encrypts data with the usage of a public key and requires five arguments as input. The first is the length of data that should be encrypted in bytes. The second is a pointer to char array which should be encrypted. The third is a pointer to char array in which encrypted data should be stored. The fourth is a pointer to RSA structure which holds the public key. The last one is the number of padding. (OpenSSL RSA)

5.3.5 Using AES algorithm

AES is working in various modes. In that application AES works in a counter mode which allows the processing of data of varying length without any problem and worries about padding. In the counter mode it is needed to create a structure which takes care of the number of iteration. This structure consists of three data. The first is the number responsible for counting iteration. The second is a pointer to char array, which stores encrypted iteration key. The third is a pointer to char array, which stores IV.

The following code listing shows a field in the structure and the initialization of structure.

```
struct ctrState{
    unsigned int number;
    unsigned char ecount[16];
    unsigned char ivec[16];
};

void initCtrState(struct ctrState *state, const unsigned char iv[8]){
    state->number = 0;
    memset(state->ecount, 0, 16);
    memset(state->ivec + 8, 0, 8);
    memcpy(state->ivec, iv, 8);
}
```

AES is a symmetric algorithm that means it uses the same key to encrypt and decrypt data, also the function used to encrypt and decrypt data is the same:

```
AES_ctr128_encrypt(messageEncrypted, message, server->iResult, &aes_key,
controlStructure.ivec, controlStructure.ecount, &controlStructure.number);
```

AES_ctr128_encrypt is a function from OpenSSL library that encrypts/decrypts data with the usage of AES algorithm in counter mode. The function needs seven arguments as input. The first is a pointer to char array with data to encrypt/decrypt. The second is a pointer to char array in which decrypted /encrypted data should be stored. The third is the length of data to encrypt/decrypt in bytes. The fourth is pointer to AES_KEY structure which stores AES rounds keys. The last three are pointers to variables from “ctrState” to control iteration of AES. (Fossies AES)

5.3.6 HMAC

HMAC is used to authenticate the source from which a message comes from. It is always created before sending the message via network and it is also send before exact message. When the application receives a message it creates hash from that message and compares it to the previously received hash. If the hashes are the same it means that data were not changed by unauthorized people.

The following code listing shows the functions which are used to create hash from a message.

```
HMAC(EVP_sha256(), ciphering->getOtherAESKey(), AES_KEYLENGTH/8 ,messageEncrypted,
server->iResult, NULL, NULL);
```

HMAC is delivered by OpenSSL library. It requires seven arguments as input. The first is name of hash function. The second is pointer to char array that stores the key which will be used to cipher data. The third is the length in bytes of key. The fourth is a pointer to char array which stores data from which

hash should be created. The fifth is the length in bytes of data to be hashed. The last two are not used in that application. (OpenSSL HMAC)

5.4 User interface layer

UI was created using Java programming language and Swing library. It consists of one application window which has two text areas for showing received and sent data. There is also a text field to input data which should be sent to another user and a button by pressing which the user confirms sending a message. At the beginning of the application those fields and the button are hidden. User has to fill in the first port number for server and IP address and the port number for client, and confirm the parameters by pressing corresponding buttons.

5.4.1 Sockets

To communicate with C++ application sockets are necessary. There is one Socket and one ServerSocket for server side and one Socket for client side. Connection is established with usage of port number 8081 for server side and port number 8080 for the client side. Communication takes place with IP address 127.0.0.1 which is consider to be loopback and allow application to send data between the applications running on the same host with usage of ports.

The following code listing shows how to create and configure sockets for UI.

```
private Socket clientSocket;
private ServerSocket server;
private Socket clientForServer;
clientSocket = new Socket("127.0.0.1", 8080);
server = new ServerSocket(8081);
clientForServer = server.accept();
```

5.4.2 Data taken from user

As it was mentioned before, at the start of the application user has to input port number for server, which should be between 1023 and 65535 and confirm this by pressing button "Turn server" to enable a receiving message from other host. Port number between 1023 and 65535 and IP address in dot-

decimal notation of host to which user want to send data should be set in field above button “Connect”, and confirmed by pressing that button (FIGURE 3).

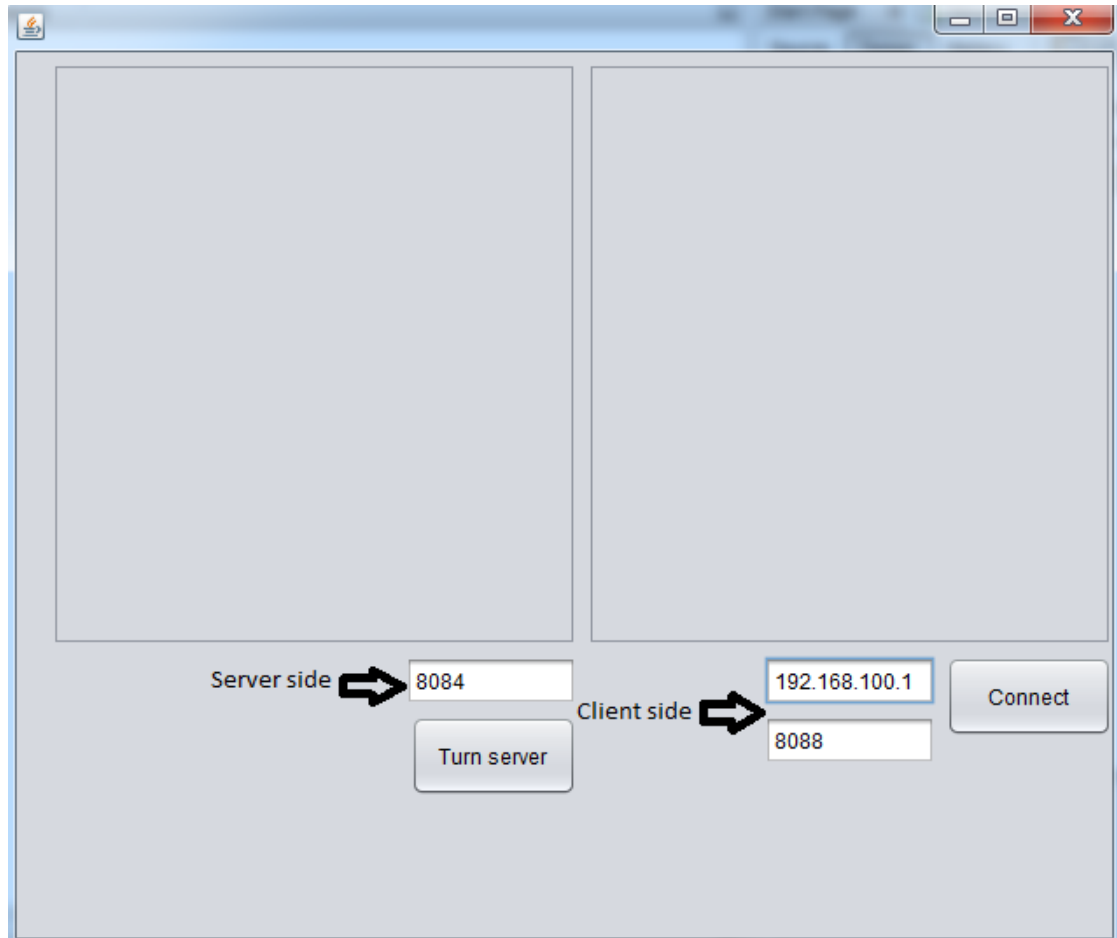


FIGURE 5 Example configurations.

After those steps user is able to send and receive a message. This step does not have to be done in that order, however, server on host with which user wants to communicate should be turned on before user sets up his own client. However, if user wants only to send or receive message, only client or server can be set up in his application.

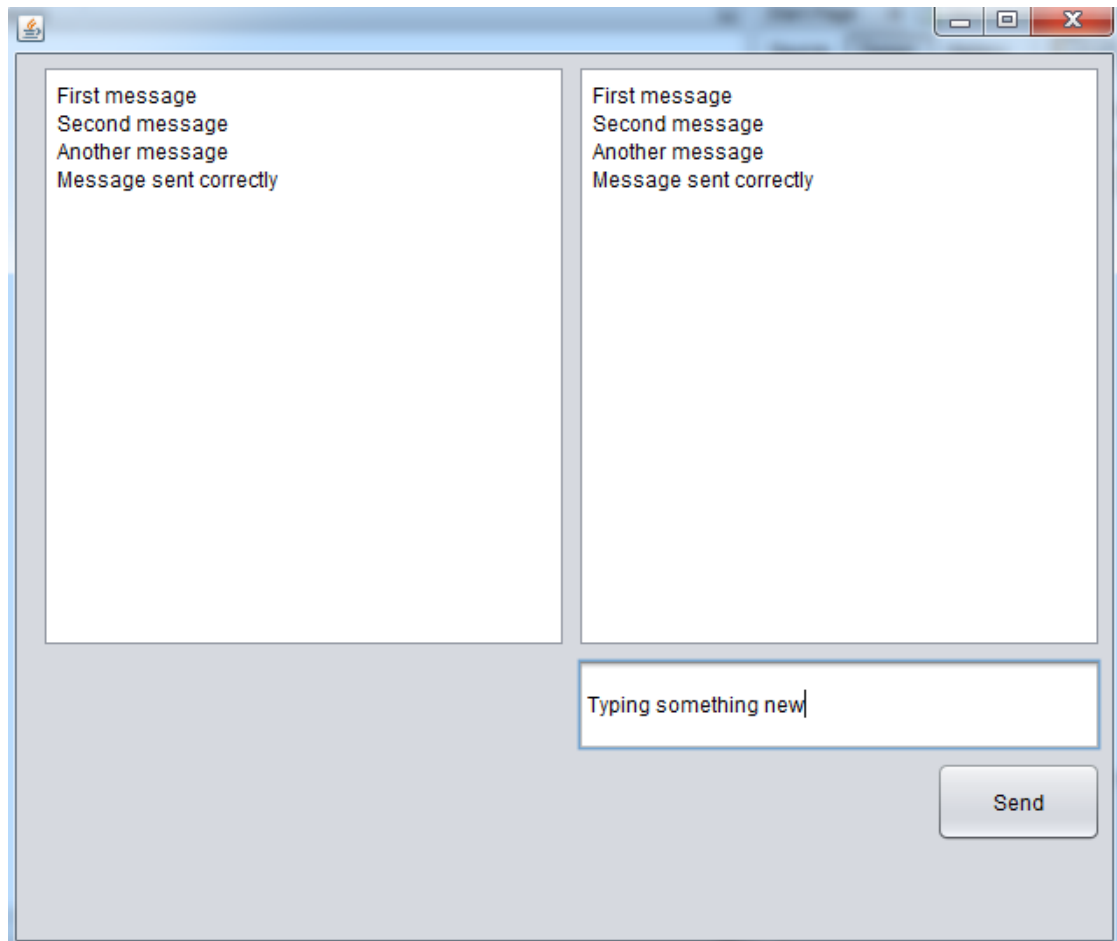


FIGURE 6 Example message exchange.

5.4.3 How to change UI

The core of “SecureCA” application was designed the way that allows to easily change UI to another. During the implementation of the new UI few things had to be kept in mind. There are two different C++ applications. “SecureCAClient” is responsible for sending message to another host, while “SecureCASServer” is responsible for receiving message. Those two C++ applications have to be started from UI.

Communication between UI and C++ application is realized with usage of sockets. “SecureCAClient” has socket configured as server and is listening for incoming connection from UI on the port number 8080 and IP address 127.0.0.1. “SecureCASServer” has socket configured as client and is trying to connect via the port number 8081 and IP address 127.0.0.1.

For “SecureCAClient” two parameters have to be input on start. The first one is the port number through which application should connect to server. The second one is the IP address of host with which communication should be established.

For “SecureCASServer” only one parameter has to be input on start. This parameter is the port number on which server will be listening for incoming connection.

6 INSTALLATION

Requirements from host:

1. Operating system : Windows 7 x 86
2. Java JDK 8:
<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
3. Win32 OpenSSL v1.0.1f and Visual C++ 2008 Redistributables:
<http://slproweb.com/products/Win32OpenSSL.html>

The program should unzipped to: "C:/SecureCA/".

After unzipping, the application should be turned on and off without establishing any connection. After that RSA key pair will be created in file:

- privateRSAKey.pem – private RSA key
- publicRSAKey.pem – public RSA key

Public RSA key should be delivered to host with which we want to communicate, and placed in folder with installed application.

After completing these steps, the application is ready to use.

7 TESTING

The test was realized with the usage of sniffers called “Wireshark” and “RawCap”. These sniffers are available for free on the following web sites:

<http://www.wireshark.org/download.html>

<http://www.netresec.com/?page=RawCap>

RawCap allows monitoring the network layer. With the usage of RawCap it is possible to check what data were sent and received by host. Wireshark was used to read the package captured by RawCap.

SecureCA connection realized with usage of TCP protocol. Message sent via this protocol was checked while application was working with RawCap. The testing was completed on one machine and the message was sent on the IP address 127.0.0.1 which is a loopback address in Windows and allows sending the message from the host to himself.

Figure 7 shows that RawCap is working. Figure 8 shows the messages sent from UI to C++ app and that data are not encrypted while on Figure 9 it is possible to see data sent from one C++ app to another and that data are encrypted. Figure 10 shows encrypted messages received from C++ application which are sent to UI.

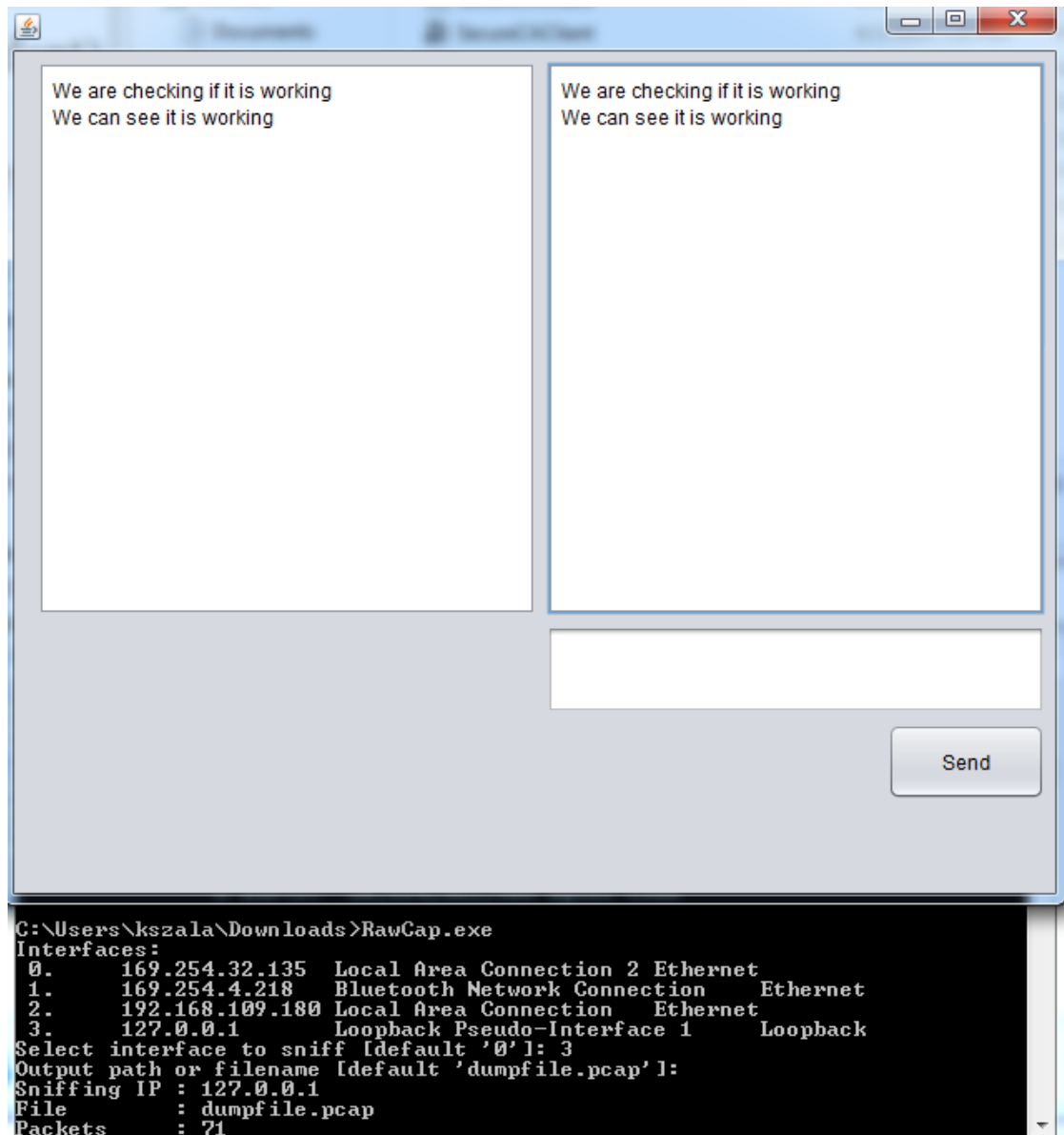


FIGURE 7 How RawCap is working.

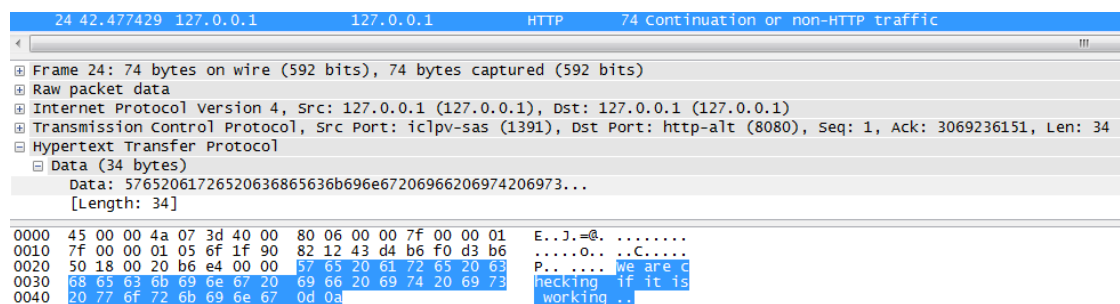


FIGURE 8 Message sent from UI to C++ application.

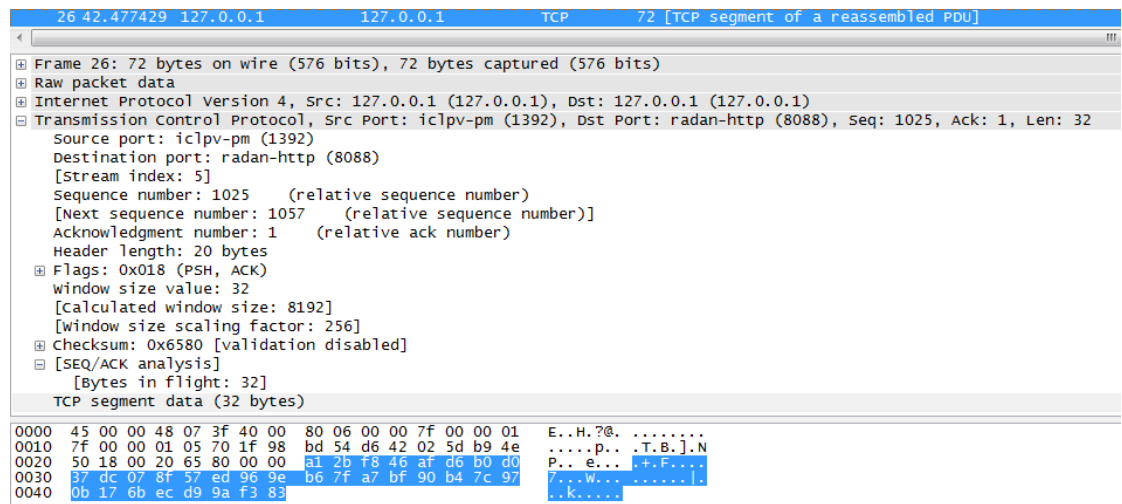


FIGURE 9 Message sent from one C++ application to another one.

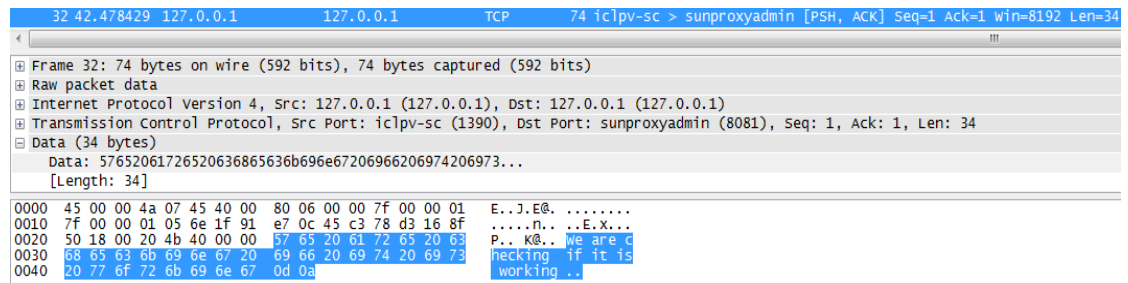


FIGURE 10 Message sent from C++ application to UI.

On Figures 8 – 10 the whole process of sending messages can be seen. The first plain text is sent from UI via the port 8080 to SecureCAClient. After that the message is encrypted and sent via the port 8088 to SecuraCASServer. At last the message is decrypted and sent via the port number 8081 to UI.

8 Further development

8.1 What has been done

All the goals that have been established have been realized. The application is working a proper way. The application is able to:

1. Generate the RSA key pair.
2. Read the RSA key from file.
3. Generate the AES key and the IV.
4. Encrypt with usage of RSA algorithm AES key and IV.
5. Establish the connection between two host via the TCP protocol
6. Send encrypted key and IV to another host.
7. Encrypt and decrypt message with usage of AES algorithm
8. Send encrypted message between two hosts.
9. Create HMAC of encrypted message.
10. Check with usage of HMAC authentication of message.
11. Communicate between UI and get from it data.

8.2 What should be done

The application still needs to be improved to provide better and more secure communication. Future development should provide:

1. Implementation of CA.
2. Communication between multiple users at one time.
3. Creating “conference” chat.
4. Better, user-friendly and intuitive UI
5. New UI should allow user to create contacts list.

It is highly recommended to create new UI with usage of C++. That will allow replacing three applications with one so the communication between UI and ciphering application will be within a single process rather than inter-process.

9 CONCLUSION

During implementing this project I learned many new issues mainly about OpenSSL. It was hard to work with that kind of library because the documentation is very poor. I already had enough knowledge about the ciphering algorithm before realization of that project. However, this project allowed me to solidify it, and even expand at some point.

Working with C++ in VS 2012 was sometimes very frustrating. However, I have learned very useful things about debugging. Implementing the communication between hosts via sockets was very simple and went for the first time without having to make any amendments. Creating UI with Java was a nice break from working with C++. Swing library is simple in usage, and creating socket communication is even simpler than in C++.

The ciphering algorithms are in my opinion one of the most important algorithms. The number of messages sent by users using the Internet continues to grow. These messages have different monetary value, however, users may want to be protected against access by third parties for personal reasons. Ciphering applications are already often used by people, and in future will be used even more often, because our society is using the internet more and more.

REFERENCES

- Fips 197 <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf> referred to, on April 3rd, 2014
- Floridi L. 2010 "Information: A Very Short Introduction" ISBN: 0199551375
- Fossies AES http://fossies.org/dox/openssl-1.0.1f/crypto_2aes_2aes_8h.html referred to, on March 30th, 2014
- MSDN Recv [http://msdn.microsoft.com/en-us/library/windows/desktop/ms740121\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms740121(v=vs.85).aspx) referred to, on March 30th, 2014
- MSDN Send [http://msdn.microsoft.com/en-us/library/windows/desktop/ms740149\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ms740149(v=vs.85).aspx) referred to, on March 30th, 2014
- OpenSSL HMAC <http://www.openssl.org/docs/crypto/hmac.html> referred to, on March 30th, 2014
- OpenSSL RSA <https://www.openssl.org/docs/crypto/rsa.html> referred to, on March 30th, 2014
- OpenSSL RAND https://www.openssl.org/docs/crypto/RAND_bytes.html referred to, on March 30th, 2014
- OpenSSL PEM <https://www.openssl.org/docs/crypto/pem.html> referred to, on March 30th, 2014
- RFC 793 <https://tools.ietf.org/html/rfc793> referred to, on April 2nd, 2014
- RFC 2014 <http://tools.ietf.org/html/rfc2104> referred to, on April 2nd, 2014
- RFC 3330 <http://tools.ietf.org/html/rfc3330> referred to, on March 30th, 2014
- RFC 6335 <https://tools.ietf.org/html/rfc6335> referred to, on March 30th, 2014
- RSA PKCS <ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf> referred to, on April 2nd, 2014
- Silicon angle <http://siliconangle.com/blog/2013/01/17/re-imagining-big-data-in-2013-predictions-for-mobile-it-and-more/> referred to, on March 30th, 2014